APPLICATION NOTE AN-00200
SDM-USB-QS-S Programmer's Guide

Introduction

The SDM-USB-QS drivers allow application software to interface with the module using calls to a DLL. The drivers take care of all of the USB protocol and timing freeing the user from the complicated firmware development. The architecture of the drivers consists of a Windows WDM driver that communicates with the device via the Windows USB Stack and a DLL that interfaces the Application Software (written in C, VC++, C++ Builder, Delphi, Visual Basic etc.) to the WDM driver. This guide documents the interface functions and gives examples of how to use them in the application software.

There are two groups of functions. First are the standard interface functions. The standard interface provides a simple, easy to use, set of functions to access the USB module. Second is the EEPROM interface, which allows the application software to read and program the various fields in the on-board EEPROM, including a user defined area that can be used for application specific purposes.

The examples of the calls will be shown in Visual Basic and C with Appendix A showing the headers and definitions for Visual Basic and Appendix B showing the same for C.

Standard Interface Functions

The standard interface functions are a series of calls made to a Dynamic Link Library (dll) that allow an application to access the module. These functions are easier to use than WIN32 API calls and offer access to features in the module for which there are no API calls.

A typical system would start with the FT_LISTDEVICES call. This call returns information about all of the modules currently connected to the bus. This allows the application software to choose which module to communicate with. Before the module can be accessed it must be opened with FT_OPEN or FT_OPENEX. These functions return a numeric handle that the rest of the functions use to identify the individual modules. Once opened, the device communications settings can be controlled. These include functions to set the baud rate (FT_SetBaudRate), set the data characteristics, such as word length, stop bits and parity (FT_SetDataCharacteristics), set hardware or software handshaking (FT_SetFlowControl), set modem control signals (FT_SetDTR, FT_ClrDTR, FT_SetRTS, FT_ClrRTS), get modem status (FT_GetModemStatus), set special characters such as event and error characters (FT_SetChars), and set receive and transmit timeouts (FT_SetTimeouts). Additional functions are available to reset the device (FT_ResetDevice), purge receive and transmit buffers (FT_Purge), get the receive queue status (FT_GetQueueStatus), get the device status (FT_GetStatus), set and reset the break condition (FT_SetBreakOn, FT_SetBreakOff), and set conditions for event notification (FT_SetEventNotification). I/O is performed using FT_Read and FT_Write. Once communications are completed, the device is closed using FT_Close.

The rest of this section will discuss these functions in detail.

Table of Contents

Introduction..... 1

Standard Interface Functions
FT_ListDevices..... 3
FT_Open..... 5
FT_OpenEx..... 6
FT_Close..... 7
FT_Read..... 8
FT_Write..... 10
FT_SetBaudRate..... 11
FT_SetDataCharacteristics..... 12
FT_SetFlowControl..... 13
FT_SetDTR..... 14
FT_ClrDTR..... 15
FT_SetRTS..... 16
FT_ClrRTS..... 17
FT_GetModemStatus..... 18
FT_SetChars..... 20
FT_Purge..... 21
FT_SetTimeouts..... 22
FT_GetQueueStatus..... 23
FT_SetBreakOn..... 24
FT_SetBreakOff..... 25
FT_GetStatus..... 26
FT_SetEventNotification..... 27
FT_ResetDevice..... 29
FT_ResetPort..... 30
FT_StopInTask..... 31
FT_RestartInTask..... 32
FT_SetResetPipeRetryCount..... 33

EEPROM Interface Functions
FT_EE_UASize..... 34
FT_EE_UARead..... 35
FT_EE_UAWrite..... 36
FT_EE_Read..... 37
FT_EE_Program..... 39
FT_EraseEE..... 41

Header Files
QS Series Visual Basic Header File..... 42
QS Series C Header File..... 45

FT_ListDevices (Arg1, Arg2, Flags)

This function can be used to return several kinds of information. First, it can be used to return the number of devices currently connected to the bus by setting Flags to LIST_NUMBER_ONLY. In this case Arg1 holds the number of devices connected to the bus and Arg2 is null. The function returns OK if successful or an error code if there is a problem.

Visual Basic

Parameter	Type	Description
Arg1	long	Holds the number of devices connected to the bus
Arg2	vbNullString	Null String
Flags	long	Constant. See the appendices for the definitions

C

Parameter	Type	Description
Arg1	pvoid	A pointer to a dword that holds the number
Arg2	Null	Null
Flags	dword	Constant. See the appendices for the definitions

This function can also be used to return the device description or serial number by setting Flags to LIST_BY_INDEX ORd with either OPEN_BY_DESCRIPTION or OPEN_BY_SERIAL_NUMBER respectively. In this case Arg1 is an integer to hold the index of the device and Arg2 is a string to hold the returned information. Indexes are zero-based and the error code DEVICE_NOT_FOUND is returned for an invalid index.

Visual Basic

Parameter	Type	Description
Arg1	integer	Holds the index number of the desired device
Arg2	string	String that holds the serial number or description
Flags	long	Constant. See the appendices for the definitions

C

Parameter	Type	Description
Arg1	dword	Holds the index number of the desired device
Arg2	char	A pointer to a buffer to contain the appropriate string
Flags	dword	Constant. See the appendices for the definitions

Examples

The following Visual Basic code demonstrates how to get the number of devices connected to the bus.

```
Dim lngStatus As Long
Dim lngNumDevices As Long

lngStatus = FT_GetNumDevices (lngNumDevices, vbNullString, LIST_NUMBER_ONLY)
If lngStatus = OK Then
        'The function was successful, the number of devices connected is in lngNumDevices
Else
        'The function failed. The error code can be reviewed and appropriate corrective action taken
End If
```

This example shows how to get the description and serial number of the first device on the bus.

```
Dim intIndex As Integer
Dim strDescription As String * 256
Dim strSerialNumber As String * 256

intIndex = 0
' Get the device description
lngStatus = FT_ListDevices (intIndex, strDescription, LIST_BY_INDEX Or OPEN_BY_DESCRIPTION)
If lngStatus = OK Then
        'The function was successful, the description is in strDescription
Else
        'The function failed. The error code can be reviewed and appropriate corrective action taken
End If

' Get the device serial number
lngStatus = FT_ListDevices (intIndex, strSerialNumber, LIST_BY_INDEX Or OPEN_BY_SERIAL_NUMBER)
If lngStatus = OK Then
        'The function was successful, the serial number is in strSerialNumber
Else
        'The function failed. The error code can be reviewed and appropriate corrective action taken
End If
```

Note that incrementing index will access the next device on the bus. If multiple devices will be connected, ListDevices can first be used to return the number of devices, then this number used to set the exit condition of a loop. The loop can increment the index and return the information for each device in turn. Following is the C code to perform the same routines as above.

```
ULONG Status;
DWORD NumDevices;

Status = FT_ListDevices (&numDevs, NULL, LIST_NUMBER_ONLY);
if (Status == OK) {
        // The function was successful, the number of devices connected is in NumDevices
}
else {
        // The function failed. The error code can be reviewed and appropriate corrective action taken
}

DWORD devIndex = 0;
char Description[256];
char SerialNumber[256];

Status = FT_ListDevices ((PVOID)devIndex, Description, LIST_BY_INDEX | OPEN_BY_DESCRIPTION);
if (Status == OK) {
        // The function was successful, the description is in Description
}
else {
        // The function failed. The error code can be reviewed and appropriate corrective action taken
}

Status = FT_ListDevices ((PVOID)devIndex, SerialNumber, LIST_BY_INDEX | OPEN_BY_SERIAL_NUMBER);
if (Status == OK) {
        // The function was successful, the serial number is in SerialNumber
}
else {
        // The function failed. The error code can be reviewed and appropriate corrective action taken
}
```

FT_Open (Device, Handle)

This function opens a device and returns a numeric handle that is used by the other functions to identify the device. Device is the index number of the device to be opened and Handle is a number that the function returns to uniquely identify the device so that other functions can access it. Since the index number of the device is used to open it so there is no ability to open a specific named device, but FT_OPEN_EX can open a device using the description or serial number. If the function executes successfully then it will return OK otherwise it will return an error code.

Visual Basic

Parameter	Type	Description
Device	integer	Index number of the device to be opened
Handle	long	A number that uniquely identifies the device

C

Parameter	Type	Description
Device	integer	Index number of the device to be opened
Handle	pvoid	A pointer to a number that uniquely identifies the device

Examples

The following Visual Basic code demonstrates this function.

```
Dim lngHandle As Long
Dim lngStatus As Long

lngStatus = FT_Open (0, lngHandle);
If lngStatus = OK Then
        'The function was successful, the handle of device 0 is in lngHandle
Else
        'The function failed. The error code can be reviewed and appropriate corrective action taken
End If
```

The following C code demonstrates this function.

```
PVOID Handle;
ULONG Status;

Status = FT_Open (0, &Handle);
if (Status == OK) {
        // The function was successful, the handle of device 0 is in Handle
}
else {
        // The function failed. The error code can be reviewed and appropriate corrective action taken
}
```

FT_OpenEx (Arg1, Flags, Handle)

This function will open a specific device using either a serial number or description and return a numeric handle that is used by other functions to access the device. Arg1 will be a string that contains either the serial number or description of the device to be opened. Flags is either OPEN_BY_SERIAL_NUMBER or OPEN_BY_DESCRIPTION and determines whether the serial number or description is used. Handle is a number that the function returns to uniquely identify the device so that other functions can access it. If the function executes successfully then it will return OK otherwise it will return an error code.

Visual Basic

Parameter	Type	Description
Arg1	string	String of the description or serial number
Flags	integer	Constant. See the appendices for the definitions
Handle	long	A number that uniquely identifies the device

C

Parameter	Type	Description
Arg1	pvoid	A pointer to a null terminated string
Flags	integer	Constant. See the appendices for the definitions
Handle	pvoid	A pointer to a number that uniquely identifies the device

Examples

The following Visual Basic code demonstrates this function.

```
Dim lngHandle As Long
Dim lngStatus As Long
Dim strSerialNumber As String * 256

lngStatus = FT_OpenEx (strSerialNumber, OPEN_BY_SERIAL_NUMBER, lngHandle)
If lngStatus = OK Then
        'The function was successful, the device's handle is in lngHandle
Else
        'The function failed. The error code can be reviewed and appropriate corrective action taken
End If
```

The following C code demonstrates this function.

```
PVOID Handle;
ULONG Status;

Status = FT_OpenEx ("LT000001", OPEN_BY_SERIAL_NUMBER, &Handle);
if (Status == OK) {
        // The function was successful, the device with serial number LT000001 is open and the handle is in Handle
}
else {
        // The function failed. The error code can be reviewed and appropriate corrective action taken
}
```

FT_Close (Handle)

This function closes communication with an open device identified by Handle. If the function executes successfully then it will return OK otherwise it will return an error code.

Visual Basic

Parameter	Type	Description
Handle	long	A number that uniquely identifies the device

C

Parameter	Type	Description
Handle	pvoid	A pointer to a number that uniquely identifies the device

Examples

The following Visual Basic code demonstrates this function.

```
Dim lngHandle As Long
Dim lngStatus As Long

lngStatus = FT_Close (lngHandle)
If lngStatus = OK Then
        'The function was successful, the device is closed
Else
        'The function failed. The error code can be reviewed and appropriate corrective action taken
End If
```

The following C code demonstrates this function.

```
PVOID Handle;
ULONG Status;

Status = FT_Close (&Handle);
if (Status == OK) {
        // The function was successful, the device is closed
}
else {
        // The function failed. The error code can be reviewed and appropriate corrective action taken
}
```

FT_Read (Handle, Buffer, BytesToRead, BytesReturned)

This function reads the data available from the device. Handle is a number returned by FT_Open or FT_OpenEx, Buffer is a string or character array that receives the data read from the device, BytesToRead is the number of bytes the function should read, and BytesReturned is the actual number of bytes that were read. If the function executes successfully then it will return OK otherwise it will return an error code.

This function does not return until BytesToRead bytes have been read into the buffer. This can cause an application to hang while waiting for the function to return. There are two ways to avoid this. The first is to get the number of bytes in the device's receive queue by calling FT_GetStatus or FT_GetQueueStatus, and passing this to FT_Read as BytesToRead so that the function reads the device and returns immediately.

The second way is by specifying a timeout in a previous call to FT_SetTimeouts. FT_Read returns when the timer expires or when BytesToRead bytes have been read, whichever occurs first. If the timeout occurred, FT_Read reads the available data into the buffer and returns OK.

An application should use the function return value and BytesReturned to check the buffer. If the return value is OK and BytesReturned is equal to BytesToRead then FT_Read has completed successfully. If the return value is OK and BytesReturned is less than BytesToRead then a timeout has occurred and the read has been only partially completed. Note that if a timeout occurred and no data was read, the return value is still OK.

Visual Basic

Parameter	Type	Description
Handle	long	A number that uniquely identifies the device
Buffer	string	String to hold the data read from the device
BytesToRead	long	The number of bytes to read from the device
BytesReturned	long	The number of bytes that were read from the device

C

Parameter	Type	Description
Handle	pvoid	A pointer to a number that uniquely identifies the device
Buffer	lpvoid	A pointer to a char array to hold the data read from the device
BytesToRead	dword	The number of bytes to read from the device
BytesReturned	lpdword	A pointer to a dword that gets the number of bytes read

Examples

The following Visual Basic code demonstrates this function. FT_GetStatus is called and the number of bytes available in the device is checked. If it is greater than zero, then FT_Read is called to get the data.

```
Dim lngHandle As Long
Dim lngStatus As Long
Dim lngBytesRead As Long
Dim strReadBuffer As String * 256
Dim lngRXBytes As Long
Dim lngTXBytes As Long
Dim lngEvents As Long
```

```
If FT_GetStatus (lngHandle, lngRXBytes, lngTXBytes, lngEvents) = OK Then
    If lngRXBytes > 0 Then
        lngStatus = FT_Read (lngHandle, strReadBuffer, lngRXBytes, lngBytesRead)
            If (lngStatus = OK) Then
                'The function was successful, the data is in strReadBuffer and lngBytesRead has the number of bytes read
            Else
                'The function failed. The error code can be reviewed and appropriate corrective action taken
            End If
    End If
End If
```

The following C code demonstrates this function.

```
PVOID Handle;
ULONG Status;
DWORD Event;
DWORD RxBytes;
DWORD TxBytes;
DWORD BytesReceived;
char RxBuffer[256];

FT_GetStatus (ftHandle, &RxBytes, &TxBytes, &Event);
if (RxBytes > 0) {
    Status = FT_Read (Handle, RxBuffer, RxBytes, &BytesReceived);
    if (Status == OK) {
        // The function was successful, the data is in RxBuffer and BytesReceived has the number of bytes read
    }
    else {
        // The function failed. The error code can be reviewed and appropriate corrective action taken
    }
}
```

FT_Write (Handle, Buffer, BytesToWrite, BytesWritten)

This function writes BytesToWrite bytes of Buffer to the device described by Handle and returns BytesWritten as the number of bytes that it actually wrote. If the function executes successfully then it will return OK otherwise it will return an error code.

Visual Basic

Parameter	Type	Description
Handle	long	A number that uniquely identifies the device
Buffer	string	String to hold the data to be written to the device
BytesToWrite	long	The number of bytes to write to the device
BytesWritten	long	The number of bytes that were written to the device

C

Parameter	Type	Description
Handle	pvoid	A pointer to a number that uniquely identifies the device
Buffer	lpvoid	A pointer to a char array to hold the data read from the device
BytesToWrite	dword	The number of bytes to read from the device
BytesWritten	lpdword	A pointer to a dword that gets the number of bytes read

Examples

The following Visual Basic code demonstrates this function.

```
Dim lngHandle As Long
Dim lngStatus As Long
Dim strWriteBuffer As String
Dim lngBytesWritten As Long
Dim lngBytesToWrite As Long

lngBytesToWrite = 1     'Sets the number of bytes to write to 1

lngStatus = FT_Write (lngHandle, strWriteBuffer, lngBytesToWrite, lngBytesWritten)
If lngStatus <> OK Then
    'The function failed. The error code can be reviewed and appropriate corrective action taken
End If
```

The following C code demonstrates this function.

```
PVOID Handle;
ULONG Status;
DWORD BytesToWrite;
DWORD BytesWritten;
char WriteBuffer[256];
char *Buff;

Buff = WriteBuffer;
BytesToWrite = 1;       // Sets the number of bytes to write to 1

Status = FT_Write (Handle, Buff, BytesToWrite, &BytesWritten);
if (Status != OK) {
    // The function failed. The error code can be reviewed and appropriate corrective action taken
}
```

FT_SetBaudRate (Handle, BaudRate)

This function sets the baud rate of the device described by Handle to BaudRate. If the function executes successfully then it will return OK otherwise it will return an error code.

Visual Basic

Parameter	Type	Description
Handle	long	A number that uniquely identifies the device
BaudRate	single	The baud rate in bits per second

C

Parameter	Type	Description
Handle	pvoid	A pointer to a number that uniquely identifies the device
BaudRate	dword	The baud rate in bits per second

Examples

The following Visual Basic code demonstrates this function.

```
Dim lngHandle As Long
Dim lngStatus As Long
Dim sngBaudRate As Single

sngBaudRate = 9600      '9600bps baud

lngStatus = FT_SetBaudRate (lngHandle, sngBaudRate)
If lngStatus = OK Then
    'The function was successful, the baud rate is set to sngBaudRate
Else
    'The function failed. The error code can be reviewed and appropriate corrective action taken
End If
```

The following C code demonstrates this function.

```
PVOID Handle;
ULONG Status;
DWORD BaudRate;

BaudRate = 9600;
Status = FT_SetBaudRate (Handle, BaudRate);
if (Status == OK) {
    // The function was successful, the baud rate is set to BaudRate
}
else {
    // The function failed. The error code can be reviewed and appropriate corrective action taken
}
```

FT_SetDataCharacteristics (Handle, WordLength, StopBits, Parity)

This function sets the data characteristics for the device described by Handle. It will set the stream to have WordLength number of bits in each word, StopBits number of stop bits, and Parity parity. WordLength must be either BITS_8 or BITS_7. StopBits must be either STOP_BITS_1 or STOP_BITS_2. Parity can be PARITY_NONE, PARITY_ODD, PARITY_EVEN, PARITY_MARK, or PARITY_SPACE. All of these variables are defined in the header files in the appendices. If the function executes successfully then it will return OK otherwise it will return an error code.

Visual Basic

Parameter	Type	Description
Handle	long	A number that uniquely identifies the device
WordLength	integer	A number representing the number of bits in each word
StopBits	integer	A number representing the number of stop bits in each word
Parity	integer	A number representing the type of parity used in each word

C

Parameter	Type	Description
Handle	pvoid	A pointer to a number that uniquely identifies the device
WordLength	uchar	A number representing the number of bits in each word
StopBits	uchar	A number representing the number of stop bits in each word
Parity	uchar	A number representing the type of parity used in each word

Examples

The following Visual Basic code demonstrates this function.

```
Dim lngHandle As Long
Dim lngStatus As Long
Dim intStops As Integer = STOP_BITS_1
Dim intParity As Integer = PARITY_NONE
Dim intDataBits As Integer = BITS_8

lngStatus = FT_SetDataCharacteristics (lngHandle, intDataBits, intStops, intParity)
If lngStatus = OK Then
    'The function was successful, the data is set to intDataBits data bits, intStops stop bits, and intParity parity
Else
    'The function failed. The error code can be reviewed and appropriate corrective action taken
End If
```

The following C code demonstrates this function.

```
PVOID Handle;
ULONG Status;
UCHAR WordLength = BITS_8;
UCHAR StopBits = STOP_BITS_1;
UCHAR Parity = PARITY_NONE;

Status = FT_SetDataCharacteristics (Handle, WordLength, StopBits, Parity);
if (Status == OK) {
    // The function was successful, the data is set to WordLength data bits, StopBits stop bits, and Parity parity
}
else {
    // The function failed. The error code can be reviewed and appropriate corrective action taken
}
```

FT_SetFlowControl (Handle, FlowControl, Xon, Xoff)

This function will set the flow control for the device described by Handle. FlowControl must be FLOW_NONE, FLOW_RTS_CTS, FLOW_DTR_DSR, or FLOW_XON_XOFF. All of these variables are defined in the header files in the appendices. Xon is the character used to signal XON, and Xoff is the character used to signal XOFF. These are only used if FlowControl is set to FLOW_XON_XOFF, otherwise they are set to zero or null. If the function executes successfully then it will return OK otherwise it will return an error code.

Visual Basic

Parameter	Type	Description
Handle	long	A number that uniquely identifies the device
FlowControl	single	A number representing the type of flow control
Xon	string	A character that signals XON
Xoff	string	A character that signals XOFF

C

Parameter	Type	Description
Handle	pvoid	A pointer to a number that uniquely identifies the device
FlowControl	ushort	A number representing the type of flow control
Xon	uchar	A character that signals XON
Xoff	uchar	A character that signals XOFF

Examples

The following Visual Basic code demonstrates this function.

```
Dim lngHandle As Long
Dim lngStatus As Long

lngStatus = FT_SetFlowControl (lngHandle, FLOW_NONE, 0, 0)
If lngStatus = OK Then
    'The function was successful, the flow control is set to FlowControl and the XON and XOFF characters are set
Else
    'The function failed. The error code can be reviewed and appropriate corrective action taken
End If
```

The following C code demonstrates this function.

```
PVOID Handle;
ULONG Status;
USHORT FlowControl = FT_FLOW_NONE;
UCHAR XonChar = 0;
UCHAR XoffChar = 0;

Status = FT_SetFlowControl (Handle, FlowControl, XonChar, XoffChar);
if (Status == OK) {
    // The function was successful, the flow control is set to FlowControl and the XON and XOFF characters are set
}
else {
    // The function failed. The error code can be reviewed and appropriate corrective action taken
}
```

FT_SetDTR (Handle)

This function sets the Data Terminal Ready control line. This can be used for handshaking when the flow control is set to FLOW_DTR_DSR, or it can be used to control external circuitry. If the function executes successfully then it will return OK otherwise it will return an error code.

Visual Basic

Parameter	Type	Description
Handle	long	A number that uniquely identifies the device

C

Parameter	Type	Description
Handle	pvoid	A pointer to a number that uniquely identifies the device

Examples

The following Visual Basic code demonstrates this function.

```
Dim lngHandle As Long
Dim lngStatus As Long

lngStatus = FT_SetDTR (lngHandle)
If lngStatus = OK Then
    'The function was successful, the DTR line is set
Else
    'The function failed. The error code can be reviewed and appropriate corrective action taken
End If
```

The following C code demonstrates this function.

```
PVOID Handle;
ULONG Status;

Status = FT_SetDTR (Handle);
if (Status == OK) {
    // The function was successful, the DTR line is set
}
else {
    // The function failed. The error code can be reviewed and appropriate corrective action taken
}
```

FT_ClrDTR (Handle)

This function clears the Data Terminal Ready control line. This can be used for handshaking when the flow control is set to FLOW_DTR_DSR, or it can be used to control external circuitry. If the function executes successfully then it will return OK otherwise it will return an error code.

Visual Basic

Parameter	Type	Description
Handle	long	A number that uniquely identifies the device

C

Parameter	Type	Description
Handle	pvoid	A pointer to a number that uniquely identifies the device

Examples

The following Visual Basic code demonstrates this function.

```
Dim lngHandle As Long
Dim lngStatus As Long

lngStatus = FT_ClrDTR (lngHandle)
If lngStatus = OK Then
    'The function was successful, the DTR line is cleared
Else
    'The function failed. The error code can be reviewed and appropriate corrective action taken
End If
```

The following C code demonstrates this function.

```
PVOID Handle;
ULONG Status;

Status = FT_ClrDTR (Handle);
if (Status == OK) {
    // The function was successful, the DTR line is cleared
}
else {
    // The function failed. The error code can be reviewed and appropriate corrective action taken
}
```

FT_SetRTS (Handle)

This function sets the Request To Send control line. This can be used for handshaking when the flow control is set to FLOW_RTS_CTS, or it can be used to control external circuitry. If the function executes successfully then it will return OK otherwise it will return an error code.

Visual Basic

Parameter	Type	Description
Handle	long	A number that uniquely identifies the device

C

Parameter	Type	Description
Handle	pvoid	A pointer to a number that uniquely identifies the device

Examples

The following Visual Basic code demonstrates this function.

```
Dim lngHandle As Long
Dim lngStatus As Long

lngStatus = FT_SetRTS (lngHandle)
If lngStatus = OK Then
    'The function was successful, the RTS line is set
Else
    'The function failed. The error code can be reviewed and appropriate corrective action taken
End If
```

The following C code demonstrates this function.

```
PVOID Handle;
ULONG Status;

Status = FT_SetRTS (Handle);
if (Status == OK) {
    // The function was successful, the RTS line is set
}
else {
    // The function failed. The error code can be reviewed and appropriate corrective action taken
}
```

FT_ClrRTS (Handle)

This function clears the Request To Send control line. This can be used for handshaking when the flow control is set to FLOW_RTS_CTS, or it can be used to control external circuitry. If the function executes successfully then it will return OK otherwise it will return an error code.

Visual Basic

Parameter	Type	Description
Handle	long	A number that uniquely identifies the device

C

Parameter	Type	Description
Handle	pvoid	A pointer to a number that uniquely identifies the device

Examples

The following Visual Basic code demonstrates this function.

```
Dim lngHandle As Long
Dim lngStatus As Long

lngStatus = FT_ClrRTS (lngHandle)
If lngStatus = OK Then
    'The function was successful, the RTS line is cleared
Else
    'The function failed. The error code can be reviewed and appropriate corrective action taken
End If
```

The following C code demonstrates this function.

```
PVOID Handle;
ULONG Status;

Status = FT_ClrRTS (Handle);
if (Status == OK) {
    // The function was successful, the RTS line is cleared
}
else {
    // The function failed. The error code can be reviewed and appropriate corrective action taken
}
```

FT_GetModemStatus (Handle, ModemStatus)

This function is used to determine the state of the input control lines, CTS, DSR, RI, and CDC. If the function executes successfully then it will return OK otherwise it will return an error code.

Visual Basic

Parameter	Type	Description
Handle	long	A number that uniquely identifies the device
ModemStatus	long	A variable that receives a number representing the modem status

C

Parameter	Type	Description
Handle	pvoid	A pointer to a number that uniquely identifies the device
ModemStatus	lpdword	A pointer to a dword variable that receives the modem status

Examples

The following Visual Basic code demonstrates this function.

```
Dim lngHandle As Long
Dim lngStatus As Long
Dim lngModemStatus As Long

lngStatus = FT_GetModemStatus (lngHandle, lngModemStatus)
If (lngModemStatus And MODEM_STATUS_CTS) = MODEM_STATUS_CTS Then
   'CTS is high
Else
   'CTS is low
End If
If (lngModemStatus And MODEM_STATUS_DSR) = MODEM_STATUS_DSR Then
   'DSR is high
Else
   'DSR is low
End If
If (lngModemStatus And MODEM_STATUS_DCD) = MODEM_STATUS_DCD Then
   'DCD is high
Else
   'DCD is low
End If
If (lngModemStatus And MODEM_STATUS_RI) = MODEM_STATUS_RI Then
   'RI is high
Else
   'RI is low
End If
```

The following C code demonstrates this function.

```
PVOID Handle;
ULONG Status;
DWORD ModemStatus;

Status = FT_GetModemStatus (Handle, &ModemStatus);
if ((ModemStatus & MODEM_STATUS_CTS) == MODEM_STATUS_CTS) {
   // CTS is high
}
else {
   // CTS is low
```

```
}
if ((ModemStatus & MODEM_STATUS_DSR) == MODEM_STATUS_DSR) {
    // DSR is high
}
else {
    // DSR is low
}
if ((ModemStatus & MODEM_STATUS_DCD) == MODEM_STATUS_DCD) {
    // DCD is high
}
else {
    // DCD is low
}
if ((ModemStatus & MODEM_STATUS_RI) == MODEM_STATUS_RI) {
    // RI is high
}
else {
    // RI is low
}
```

FT_SetChars (Handle, EventCh, EventChEn, ErrorCh, ErrorChEn)

This function sets the special characters for the device. If the function executes successfully then it will return OK otherwise it will return an error code.

Visual Basic

Parameter	Type	Description
Handle	long	A number that uniquely identifies the device
EventCh	string	Event character
EventChEn	string	0 if the event character is disabled, non-zero otherwise
ErrorCh	string	Error character
ErrorChEn	string	0 if the error character is disabled, non-zero otherwise

C

Parameter	Type	Description
Handle	pvoid	A pointer to a number that uniquely identifies the device
EventCh	uchar	Event character
EventChEn	uchar	0 if the event character is disabled, non-zero otherwise
ErrorCh	uchar	Error character
ErrorChEn	uchar	0 if the error character is disabled, non-zero otherwise

Examples

The following Visual Basic code demonstrates this function.

```
Dim lngHandle As Long
Dim lngStatus As Long
Dim strEventCh As String
Dim strEventChEn As String
Dim strErrorCh As String
Dim strErrorChEn As String

lngStatus = FT_SetChars (lngHandle, strEventCh, strEventChEn, strErrorCh, strErrorChEn)
If lngStatus = OK Then
    'The function was successful, the RTS line is cleared
Else
    'The function failed. The error code can be reviewed and appropriate corrective action taken
End If
```

The following C code demonstrates this function.

```
PVOID Handle;
ULONG Status;
UCHAR EventCh;
UCHAR EventChEn;
UCHAR ErrorCh;
UCHAR ErrorChEn;

Status = FT_SetChars (Handle, EventCh, EventChEn, ErrorCh, ErrorChEn);
if (Status == OK) {
   // The function was successful,
}
else {
   // The function failed. The error code can be reviewed and appropriate corrective action taken
}
```

FT_Purge (ftHandle, Mask)

This function purges receive and transmit buffers in the device. If the function executes successfully then it will return OK otherwise it will return an error code.

Visual Basic

Parameter	Type	Description
Handle	long	A number that uniquely identifies the device
Mask	long	Any combination of PURGE_RX and PURGE_TX

C

Parameter	Type	Description
Handle	pvoid	A pointer to a number that uniquely identifies the device
Mask	dword	Any combination of PURGE_RX and PURGE_TX

Examples

The following Visual Basic code demonstrates this function.

```
Dim lngHandle As Long
Dim lngStatus As Long
Dim lngMask As Long

lngMask = PURGE_RX Or PURGE_TX

lngStatus = FT_Purge (lngHandle, lngMask)
If lngStatus = OK Then
    'The function was successful, the receive and transmit buffers have been cleared
Else
    'The function failed. The error code can be reviewed and appropriate corrective action taken
End If
```

The following C code demonstrates this function.

```
PVOID Handle;
ULONG Status;
DWORD Mask;

Mask = PURGE_RX | PURGE_TX;

Status = FT_Purge (Handle, Mask);
if (Status == OK) {
    // The function was successful, the receive and transmit buffers have been cleared
}
else {
    // The function failed. The error code can be reviewed and appropriate corrective action taken
}
```

FT_SetTimeouts (Handle, ReadTimeout, WriteTimeout)

This function sets the read and write timeouts for the device. If the function executes successfully then it will return OK otherwise it will return an error code.

Visual Basic

Parameter	Type	Description
Handle	long	A number that uniquely identifies the device
ReadTimeout	long	Read timeout in milliseconds
WriteTimeout	long	Write timeout in milliseconds

C

Parameter	Type	Description
Handle	pvoid	A pointer to a number that uniquely identifies the device
ReadTimeout	dword	Read timeout in milliseconds
WriteTimeout	dword	Write timeout in milliseconds

Examples

The following Visual Basic code demonstrates this function.

```
Dim lngHandle As Long
Dim lngStatus As Long

lngStatus = FT_SetTimeouts (lngHandle, 5000, 1000)
If lngStatus = OK Then
   'The function was successful, the read timeout is set to 5 seconds and the write timeout is set to 1 second
Else
    'The function failed. The error code can be reviewed and appropriate corrective action taken
End If
```

The following C code demonstrates this function.

```
PVOID Handle;
ULONG Status;

Status = FT_SetTimeouts (Handle, 5000, 1000);
if (Status == OK) {
   // The function was successful, the read timeout is set to 5 seconds and the write timeout is set to 1 second
}
else {
   // The function failed. The error code can be reviewed and appropriate corrective action taken
}
```

FT_GetQueueStatus (Handle, AmountInRxQueue)

This function gets the number of characters currently in the receive queue and places the value in AmountInRxQueue. This function can be called and the value in AmountInRxQueue can be passed to FT_Read as BytesToRead so that the Read function will read the receive buffer and return immediately. If the function executes successfully then it will return OK otherwise it will return an error code.

Visual Basic

Parameter	Type	Description
Handle	long	A number that uniquely identifies the device
AmountInRxQueue	long	Receives the number of characters in the receive queue

C

Parameter	Type	Description
Handle	pvoid	A pointer to a number that uniquely identifies the device
AmountInRxQueue	lpdword	A pointer to a dword that gets the number of characters available

Examples

The following Visual Basic code demonstrates this function.

```
Dim lngHandle As Long
Dim lngStatus As Long
Dim lngAnountInRxQueue as Long

lngStatus = FT_GetQueueStatus (lngHandle, lngAnountInRxQueue)
If lngStatus = OK Then
   'The function was successful, the number of characters in the receive queue is in lngAnountInRxQueue
Else
    'The function failed. The error code can be reviewed and appropriate corrective action taken
End If
```

The following C code demonstrates this function.

```
PVOID Handle;
ULONG Status;
DWORD AnountInRxQueue;

Status = FT_GetQueueStatus (Handle, &AnountInRxQueue);
if (Status == OK) {
   // The function was successful, the number of characters in the receive queue is in AnountInRxQueue
}
else {
   // The function failed. The error code can be reviewed and appropriate corrective action taken
}
```

FT_SetBreakOn (Handle)

This function sets the break condition for the device. If the function executes successfully then it will return OK otherwise it will return an error code.

Visual Basic

Parameter	Type	Description
Handle	long	A number that uniquely identifies the device

C

Parameter	Type	Description
Handle	pvoid	A pointer to a number that uniquely identifies the device

Examples

The following Visual Basic code demonstrates this function.

```
Dim lngHandle As Long
Dim lngStatus As Long

lngStatus = FT_SetBreakOn (lngHandle)
If lngStatus = OK Then
   'The function was successful, the break condition is set
Else
    'The function failed. The error code can be reviewed and appropriate corrective action taken
End If
```

The following C code demonstrates this function.

```
PVOID Handle;
ULONG Status;

Status = FT_SetBreakOn (Handle);
if (Status == OK) {
   // The function was successful, the break condition is set
}
else {
   // The function failed. The error code can be reviewed and appropriate corrective action taken
}
```

FT_SetBreakOff (Handle)

This function resets the break condition for the device. If the function executes successfully then it will return OK otherwise it will return an error code.

Visual Basic

Parameter	Type	Description
Handle	long	A number that uniquely identifies the device

C

Parameter	Type	Description
Handle	pvoid	A pointer to a number that uniquely identifies the device

Examples

The following Visual Basic code demonstrates this function.

```
Dim lngHandle As Long
Dim lngStatus As Long

lngStatus = FT_SetBreakOff (lngHandle)
If lngStatus = OK Then
   'The function was successful, the break condition is reset
Else
   'The function failed. The error code can be reviewed and appropriate corrective action taken
End If
```

The following C code demonstrates this function.

```
PVOID Handle;
ULONG Status;

Status = FT_SetBreakOff (Handle);
if (Status == OK) {
   // The function was successful, the break condition is reset
}
else {
   // The function failed. The error code can be reviewed and appropriate corrective action taken
}
```

FT_GetStatus (Handle, AmountInRxQueue, AmountInTxQueue, EventStatus)

This function gets the status of the device. AmountInRxQueue gets the number of characters in the receive queue, AmountInTxQueue gets the number of characters in the transmit queue, and EventStatus gets a combination of EVENT_RXCHAR if a character is received and EVENT_MODEM_STATUS if the modem lines change states. If the function executes successfully then it will return OK otherwise it will return an error code.

Visual Basic

Parameter	Type	Description
Handle	long	A number that uniquely identifies the device
AmountInRxQueue	long	Gets the number of bytes in the receive queue
AmountInTxQueue	long	Gets the number of bytes in the transmit queue
EventStatus	long	Gets a value of an event or returns zero

C

Parameter	Type	Description
Handle	pvoid	A pointer to a number that uniquely identifies the device
AmountInRxQueue	lpdword	A pointer to a dword that gets the amount in the receive queue
AmountInTxQueue	lpdword	A pointer to a dword that gets the amount in the transmit queue
EventStatus	lpdword	A pointer to a dword that gets the event status

Examples

The following Visual Basic code demonstrates this function.

```
Dim lngHandle As Long
Dim lngStatus As Long
Dim lngAmountInRxQueue As Long
Dim lngAmountInTxQueue As Long
Dim lngEventStatus As Long

lngStatus = FT_GetStatus (lngHandle, lngAmountInRxQueue, lngAmountInTxQueue, lngEventStatus)
If lngStatus = OK Then
   'The function was successful
Else
    'The function failed. The error code can be reviewed and appropriate corrective action taken
End If
```

The following C code demonstrates this function.

```
PVOID Handle;
ULONG Status;
DWORD AmountInRxQueue;
DWORD AmountInTxQueue;
DWORD EventStatus;

Status = FT_GetStatus (Handle, &AmountInRxQueue, &AmountInTxQueue, &EventStatus);
if (Status == OK) {
   // The function was successful
}
else {
   // The function failed. The error code can be reviewed and appropriate corrective action taken
}
```

FT_SetEventNotification (Handle, EventMask, Arg)

This function will set the events that the device should look for. EventMask is any combination of EVENT_RXCHAR and EVENT_MODEM_STATUS. EVENT_RXCHAR will cause the event to be set when a character has been received, and EVENT_MODEM_STATUS will cause the event to be set when the modem lines change. Arg is the handle of an event that has been created by the application. This function can be used by an application to set up conditions that allow a thread to block until one of the conditions is met. Typically, an application will create an event, call this function, then block on the event. When the conditions are met, the event is set, and the application thread unblocked. If the function executes successfully then it will return OK otherwise it will return an error code.

Visual Basic

Parameter	Type	Description
Handle	long	A number that uniquely identifies the device
EventMask	long	Bit map describing the conditions that cause the event to be set
Arg	long	The handle of an event

C

Parameter	Type	Description
Handle	pvoid	A pointer to a number that uniquely identifies the device
EventMask	dword	Bit map describing the conditions that cause the event to be set
Arg	pvoid	The handle of an event

Examples

The following Visual Basic code demonstrates this function. First, an event is created and the function is called.

```
Const INFINITE As Long = 1000   '&HFFFFFFFF
Dim lngHandle As Long
Dim lngStatus As Long
Dim lngEventMask As Long
Dim lngEvent As Long
Dim lngModemStatus As Long

lngEvent = CreateEvent (0, False, False, "")
lngEventMask = EVENT_RXCHAR Or EVENT_MODEM_STATUS
lngStatus = FT_SetEventNotification (lngHandle, lngEventMask, lngEvent)

   'This will wait for the event to trigger and release the object
lngStatus = WaitForSingleObject (lngEvent, INFINITE)

   'Call FT_GetModemStatus to determine what caused the event
lngStatus = FT_GetModemStatus (lngHandle, lngModemStatus)
If (lngModemStatus And MODEM_STATUS_CTS) = MODEM_STATUS_CTS Then
   'CTS is high
Else
   'CTS is low
End If
If (lngModemStatus And MODEM_STATUS_DSR) = MODEM_STATUS_DSR Then
   'DSR is high
Else
   'DSR is low
```

```
End If
If (lngModemStatus And MODEM_STATUS_DCD) = MODEM_STATUS_DCD Then
   'DCD is high
Else
   'DCD is low
End If
If (lngModemStatus And MODEM_STATUS_RI) = MODEM_STATUS_RI Then
   'RI is high
Else
   'RI is low
End If
```

The following C code demonstrates this function. First, an event is created and the function is called.

```
#define INFINITE    1000
PVOID Handle;
ULONG Status;
PVOID Event;
DWORD EventMask;
DWORD ModemStatus;

Event = CreateEvent (NULL, false, false, "" );
EventMask = EVENT_RXCHAR | EVENT_MODEM_STATUS;
Status = FT_SetEventNotification (Handle, EventMask, Event);

   // This will wait for the event to trigger and release the object
WaitForSingleObject (Event, INFINITE);

   // Call FT_GetModemStatus to determine what caused the event
Status = FT_GetModemStatus (Handle, &ModemStatus);
if ((ModemStatus & MODEM_STATUS_CTS) == MODEM_STATUS_CTS) {
    // CTS is high
}
else {
    // CTS is low
}
if ((ModemStatus & MODEM_STATUS_DSR) == MODEM_STATUS_DSR) {
    // DSR is high
}
else {
    // DSR is low
}
if ((ModemStatus & MODEM_STATUS_DCD) == MODEM_STATUS_DCD) {
    // DCD is high
}
else {
    // DCD is low
}
if ((ModemStatus & MODEM_STATUS_RI) == MODEM_STATUS_RI) {
    // RI is high
}
else {
    // RI is low
}
```

FT_ResetDevice (Handle)

This function will reset the device described by Handle. If the function executes successfully then it will return OK otherwise it will return an error code.

Visual Basic

Parameter	Type	Description
Handle	long	A number that uniquely identifies the device

C

Parameter	Type	Description
Handle	pvoid	A pointer to a number that uniquely identifies the device

Examples

The following Visual Basic code demonstrates this function.

```
Dim lngHandle As Long
Dim lngStatus As Long

lngStatus = FT_ResetDevice (lngHandle)
If lngStatus = OK Then
   'The function was successful, the device is reset
Else
    'The function failed. The error code can be reviewed and appropriate corrective action taken
End If
```

The following C code demonstrates this function.

```
PVOID Handle;
ULONG Status;

Status = FT_ResetDevice (Handle);
if (Status == OK) {
   // The function was successful, the device is reset
}
else {
   // The function failed. The error code can be reviewed and appropriate corrective action taken
}
```

FT_ResetPort (Handle)

This function will send a reset command to the port in an attempt to recover the port after a failure. If the function executes successfully then it will return OK otherwise it will return an error code.

Visual Basic

Parameter	Type	Description
Handle	long	A number that uniquely identifies the device

C

Parameter	Type	Description
Handle	pvoid	A pointer to a number that uniquely identifies the device

Examples

The following Visual Basic code demonstrates this function.

```
Dim lngHandle As Long
Dim lngStatus As Long

lngStatus = FT_ResetPort (lngHandle)
If lngStatus = OK Then
   'The function was successful, the port is reset
Else
   'The function failed. The error code can be reviewed and appropriate corrective action taken
End If
```

The following C code demonstrates this function.

```
PVOID Handle;
ULONG Status;

Status = FT_ResetPort (Handle);
if (Status == OK) {
   // The function was successful, the port is reset
}
else {
   // The function failed. The error code can be reviewed and appropriate corrective action taken
}
```

FT_StopInTask (Handle)

This function is used to put the driver's IN task (read) into a wait state. It can be used in situations where data is being received continuously so that the device can be purged without more data being received. It is used together with FT_RestartInTask, which sets the IN task running again. If the function executes successfully then it will return OK otherwise it will return an error code.

Visual Basic

Parameter	Type	Description
Handle	long	A number that uniquely identifies the device

C

Parameter	Type	Description
Handle	pvoid	A pointer to a number that uniquely identifies the device

Examples

The following Visual Basic code demonstrates this function.

```
Dim lngHandle As Long
Dim lngStatus As Long

Do
   lngStatus = FT_StopInTask (lngHandle)
Loop While lngStatus <> OK

'Do something, for example purge the device

Do
   lngStatus = FT_RestartInTask (lngHandle)
Loop While lngStatus <> OK
```

The following C code demonstrates this function.

```
PVOID Handle;
ULONG Status;

do {
        Status = FT_StopInTask (Handle);
} while (Status != OK);

// Do something, for example purge device

do {
        Status = FT_RestartInTask (Handle);
} while (Status != OK);
```

FT_RestartInTask (Handle)

This function is used to restart the driver's IN task (read) after it has been stopped by a call to FT_StopInTask. If the function executes successfully then it will return OK otherwise it will return an error code.

Visual Basic

Parameter	Type	Description
Handle	long	A number that uniquely identifies the device

C

Parameter	Type	Description
Handle	pvoid	A pointer to a number that uniquely identifies the device

Examples

The following Visual Basic code demonstrates this function.

```
Dim lngHandle As Long
Dim lngStatus As Long

Do
   lngStatus = FT_StopInTask (lngHandle)
Loop While lngStatus <> OK

'Do something, for example purge device

Do
   lngStatus = FT_RestartInTask (lngHandle)
Loop While lngStatus <> OK
```

The following C code demonstrates this function.

```
PVOID Handle;
ULONG Status;

do {
        Status = FT_StopInTask (Handle);
} while (Status != OK);

// Do something, for example purge device

do {
        Status = FT_RestartInTask (Handle);
} while (Status != OK);
```

FT_SetResetPipeRetryCount (Handle, Count)

This function is used to set the ResetPipeRetryCount. ResetPipeRetryCount controls the maximum number of times that the driver tries to reset a pipe on which an error has occurred. ResetPipeRequestRetryCount defaults to 50. It may be necessary to increase this value in noisy environments where a lot of USB errors occur. If the function executes successfully then it will return OK otherwise it will return an error code.

Visual Basic

Parameter	Type	Description
Handle	long	A number that uniquely identifies the device
Count	long	Contains the maximum number of times to try to reset the pipe

C

Parameter	Type	Description
Handle	pvoid	A pointer to a number that uniquely identifies the device
Count	dword	Contains the maximum number of times to try to reset the pipe

Examples

The following Visual Basic code demonstrates this function.

```
Dim lngHandle As Long
Dim lngStatus As Long
Dim lngRetryCount As Long

lngRetryCount = 100
lngStatus = FT_SetResetPipeRetryCount (lngHandle, lngRetryCount)
If lngStatus = OK Then
   'The function was successful, ResetPipeRetryCount is set to 100
Else
   'The function failed. The error code can be reviewed and appropriate corrective action taken
End If
```

The following C code demonstrates this function.

```
PVOID Handle;
ULONG Status;
DWORD RetryCount;

RetryCount = 100;
Status = FT_SetResetPipeRetryCount (Handle, RetryCount);
if (Status == OK) {
        // The function was successful, ResetPipeRetryCount is set to 100
}
else {
        // The function failed. The error code can be reviewed and appropriate corrective action taken
}
```

EEPROM Interface Functions

The EEPROM interface functions allow the application to access the on-board EEPROM. This can be useful in production to allow the programming of the device description as a part of the final production test. In addition, the application can use the free area to store a small amount of information.

FT_EE_UASize (Handle, Size)

This function determines the size of the User Area in the EEPROM and returns the number of bytes free in Size. This is the largest amount of data that can be stored in the EEPROM by the application. If the function executes successfully then it will return OK otherwise it will return an error code.

Visual Basic

Parameter	Type	Description
Handle	long	A number that uniquely identifies the device
Size	long	A variable that gets the size of the free area in bytes

C

Parameter	Type	Description
Handle	pvoid	A pointer to a number that uniquely identifies the device
Size	lpdword	A pointer to a variable that gets the size of the free area in bytes

Examples

The following Visual Basic code demonstrates this function.

```
Dim lngHandle As Long
Dim lngStatus As Long
Dim lngSize As Long

lngStatus = FT_EE_UASize (lngHandle, lngSize)
If lngStatus = OK Then
   'The function was successful, Size contains the number of bytes free in the EEPROM
Else
    'The function failed. The error code can be reviewed and appropriate corrective action taken
End If
```

The following C code demonstrates this function.

```
PVOID Handle;
ULONG Status;
DWORD Size;

Status = FT_EE_UASize (Handle, &Size);
if (Status == OK) {
        // The function was successful, Size contains the number of bytes free in the EEPROM
}
else {
        // The function failed. The error code can be reviewed and appropriate corrective action taken
}
```

FT_EE_UARead (Handle, Data, DataLen, BytesRead)

This function will read the data in the User Area on the EEPROM. Data contains the data that was read by the function, DataLen is the size of the string or character array that receives the data, and BytesRead is the actual number of bytes that were read. If DataLen is less than the size of the UA, then only DataLen bytes are read into the buffer. Otherwise, the entire UA is read into the buffer. If the function executes successfully then it will return OK otherwise it will return an error code.

Visual Basic

Parameter	Type	Description
Handle	long	A number that uniquely identifies the device
Data	string	A buffer that holds the data from the User Area
DataLen	long	A variable that holds the number of bytes to be read from the UA
BytesRead	long	A variable that holde the actual number of bytes read from the UA

C

Parameter	Type	Description
Handle	pvoid	A pointer to a number that uniquely identifies the device
Data	puchar	A pointer to a buffer that holds the data from the User Area
DataLen	dword	A variable that holds the number of bytes to be read from the UA
BytesRead	lpdword	A pointer to a variable that receives teh number of bytes read

Examples

The following Visual Basic code demonstrates this function.

```
Dim lngHandle As Long
Dim lngStatus As Long
Dim strData As String * 64
Dim lngDataLen As Long
Dim lngBytesRead As Long

lngDataLen = 64
lngStatus = FT_EE_UARead (lngHandle, strData, lngDataLen, lngBytesRead)
If lngStatus = OK Then
   'The function was successful, strData holds lngBytesRead bytes of data read from the UA on the EEPROM
Else
   'The function failed. The error code can be reviewed and appropriate corrective action taken
End If
```

The following C code demonstrates this function.

```
PVOID Handle;
ULONG Status;
CHAR Data[64];
DWORD DataLen;
DWORD BytesRead;

DataLen = 64;
Status = FT_EE_UARead (Handle, Data, DataLen, &BytesRead);
if (Status == OK) {
        // The function was successful, Data holds BytesRead bytes of data read from the UA on the EEPROM
}
else {
        // The function failed. The error code can be reviewed and appropriate corrective action taken
}
```

FT_EE_UAWrite (Handle, Data, DataLen)

This function will write the information contained in Data to the User Area in the EEPROM. DataLen contains the amount of data to be written. If the function executes successfully then it will return OK otherwise it will return an error code.

Visual Basic

Parameter	Type	Description
Handle	long	A number that uniquely identifies the device
Data	string	A buffer that holds the data from the User Area
DataLen	long	A variable that holds the number of bytes to be read from the UA

C

Parameter	Type	Description
Handle	pvoid	A pointer to a number that uniquely identifies the device
Data	puchar	A pointer to a buffer that holds the data from the User Area
DataLen	dword	A variable that holds the number of bytes to be read from the UA

Examples

The following Visual Basic code demonstrates this function.

```
Dim lngHandle As Long
Dim lngStatus As Long
Dim strData As String * 64
Dim lngDataLen As Long

lngDataLen = 64
lngStatus = FT_EE_UAWrite (lngHandle, strData, lngDataLen)
If lngStatus = OK Then
   'The function was successful, the UA on the EEPROM contains strData
Else
    'The function failed. The error code can be reviewed and appropriate corrective action taken
End If
```

The following C code demonstrates this function.

```
PVOID Handle;
ULONG Status;
CHAR Data[64];
DWORD DataLen;

DataLen = 64;
Status = FT_EE_UAWrite (Handle, &Data, DataLen);
if (Status == OK) {
        // The function was successful, the UA on the EEPROM contains Data
}
else {
        // The function failed. The error code can be reviewed and appropriate corrective action taken
}
```

FT_EE_Read (Handle, Data)

This function will read the contents of the programmed section of the EEPROM and place the information into data structure Data. The type definition for Data is included in the header files at the end of this document. The function does not perform any checks on buffer sizes, so the buffers passed in the PROGRAM_DATA structure must be big enough to accommodate their respective strings (including null terminators). The sizes shown in the following example are more than adequate and can be rounded down if necessary. The restriction is that the Manufacturer string length plus the Description string length is less than or equal to 40 characters. If the function executes successfully then it will return OK otherwise it will return an error code.

Visual Basic

Parameter	Type	Description
Handle	long	A number that uniquely identifies the device
Data	structure	A structure of type PROGRAM_DATA

C

Parameter	Type	Description
Handle	pvoid	A pointer to a number that uniquely identifies the device
Data	structure	A pointer to a structure of type PROGRAM_DATA

Examples

Using this function in Visual Basic becomes complicated because the PROGRAM_DATA structure contains only POINTERS to bytearrays. This means that the variables Manufacturer, ManufacturerID, Description and SerialNumber are passed as POINTERS to the locations of bytearrays. Each Byte in these arrays will be filled with one character of the whole string. Visual Basic supports getting the addresses of pointers, however the functions to do so are undocumented. For more information on how to get pointers to variables in Visual Basic, see Microsoft Knowledge Base Article Q199824. The function used in this example is VarPtr, which returns the address of a variable. The following Visual Basic code demonstrates this function.

```
Dim lngHandle As Long
Dim lngStatus As Long
Dim EEData As PROGRAM_DATA

'Bytearrays as "string-containers":
Dim bManufacturer(32) As Byte
Dim bManufacturerID(16) As Byte
Dim bDescription(64) As Byte
Dim bSerialNumber(16) As Byte

'Use an undocumented function to return a pointer
EEData.Manufacturer = VarPtr (bManufacturer(0))
EEData.ManufacturerId = VarPtr (bManufacturerID(0))
EEData.Description = VarPtr (bDescription(0))
EEData.SerialNumber = VarPtr (bSerialNumber(0))

lngStatus = FT_EE_Read (lngHandle, EEData)
If lngStatus = OK Then
   'The function was successful, the information in the EEPROM is in EEData
   'Convert the resulting bytearrays to strings (NULL-characters at the end are cut off)
   strManufacturer = StrConv (bManufacturer, vbUnicode)
   strManufacturer = Left (strManufacturer, InStr (strManufacturer, Chr(0)) - 1)
```

```
    strManufacturerID = StrConv (bManufacturerID, vbUnicode)
    strManufacturerID = Left (strManufacturerID, InStr (strManufacturerID, Chr(0)) - 1)

    strDescription = StrConv (bDescription, vbUnicode)
    strDescription = Left (strDescription, InStr (strDescription, Chr(0)) - 1)

    strSerialNumber = StrConv (bSerialNumber, vbUnicode)
    strSerialNumber = Left (strSerialNumber, InStr (strSerialNumber, Chr(0)) - 1)
Else
    'The function failed. The error code can be reviewed and appropriate corrective action taken
End If
```

The following C code demonstrates this function.

```
PVOID Handle;
ULONG Status;
PROGRAM_DATA EEData;
char ManufacturerBuf[32];
char ManufacturerIdBuf[16];
char DescriptionBuf[64];
char SerialNumberBuf[16];

EEData.Manufacturer = ManufacturerBuf;
EEData.ManufacturerId = ManufacturerIdBuf;
EEData.Description = DescriptionBuf;
EEData.SerialNumber = SerialNumberBuf;

Status = FT_EE_Read (Handle, &EEData);
if (Status == OK) {
        // The function was successful, the information in the EEPROM is in EEData
}
else {
        // The function failed. The error code can be reviewed and appropriate corrective action taken
}
```

FT_EE_Program (Handle, Data)

This function will write the contents of structure Data to the EEPROM. The type definition for Data is included in the header files at the end of this document. If the SerialNumber field in PROGRAM_DATA is NULL, or SerialNumber points to a NULL string, a serial number based on the ManufacturerId and the current date and time will be generated. If the function executes successfully then it will return OK otherwise it will return an error code.

Visual Basic

Parameter	Type	Description
Handle	long	A number that uniquely identifies the device
Data	structure	A structure of type PROGRAM_DATA

C

Parameter	Type	Description
Handle	pvoid	A pointer to a number that uniquely identifies the device
Data	structure	A pointer to a structure of type PROGRAM_DATA

Examples

Using this function in Visual Basic becomes complicated because the PROGRAM_DATA structure contains only POINTERS to bytearrays. This means that the variables Manufacturer, ManufacturerID, Description and SerialNumber are passed as POINTERS to the locations of bytearrays. Each Byte in these arrays will be filled with one character of the whole string. Visual Basic supports getting the addresses of pointers, however the functions to do so are undocumented. For more information on how to get pointers to variables in Visual Basic, see Microsoft Knowledge Base Article Q199824. The function used in this example is VarPtr, which returns the address of a variable. The following Visual Basic code demonstrates this function.

```
Dim lngHandle As Long
Dim lngStatus As Long
Dim EEData As PROGRAM_DATA
Dim strManufacturer As String
Dim strManufacturerID As String
Dim strDescription As String
Dim strSerialNumber As String

'Declare byte arrays as "string-containers":
Dim bManufacturer(32) As Byte
Dim bManufacturerID(16) As Byte
Dim bDescription(64) As Byte
Dim bSerialNumber(16) As Byte

'Load the strings
strManufacturer = "Linx Technologies"
strManufacturerID = "LT"
strDescription = "LINX SDM-USB-QS-S"
strSerialNumber = ""

'Load the EEData structure with the default data
EEData.VendorId = 0x0403
EEData.ProductId = 0xF448
EEData.MaxPower = 100
EEData.PnP = 1
EEData.SelfPowered = 0
```

```
EEData.RemoteWakeup = 1
EEData.Rev4 = TRUE
EEData.IsoIn = TRUE
EEData.IsoOut = TRUE
EEData.PullDownEnable = TRUE
EEData.SerNumEnable = FALSE
EEData.USBVersionEnable = FALSE
EEData.USBVersion = 0

'Use an undocumented function to return a pointer
EEData.Manufacturer = VarPtr(bManufacturer(0))
EEData.ManufacturerId = VarPtr(bManufacturerID(0))
EEData.Description = VarPtr(bDescription(0))
EEData.SerialNumber = VarPtr(bSerialNumber(0))

'Convert the strings to byte arrays
StringToByteArray (strManufacturer, bManufacturer)
StringToByteArray (strManufacturerID, bManufacturerID)
StringToByteArray (strDescription, bDescription)
StringToByteArray (strSerialNumber, bSerialNumber)

'Now write the complete set of EEPROM data
lngStatus = FT_EE_Program (lngHandle, EEData)
If lngStatus = OK Then
   'The function was successful, the information in the EEPROM is in EEData
Else
    'The function failed. The error code can be reviewed and appropriate corrective action taken
End If

'This function will convert a string to a byte array
Private Sub StringToByteArray (strString, bByteArray)
Dim lngN As Long
'Fill bByteArray with "0":
For lngN = 0 To UBound (bByteArray)
   bByteArray (lngN) = 0
Next
For lngN = 1 To Len(strString)
   bByteArray(lngN - 1) = Asc(Mid(strString, lngN, 1))
Next
End Sub
```

The following C code demonstrates this function.

```
PVOID Handle;
ULONG Status;
PROGRAM_DATA EEData;

// Load the EEData structure with the default data
EEData = {0x0403, 0xF449, "Linx Technologies", "LT", "LINX SDM-USB-QS-S", "", 44, 1, 0, 1, TRUE, TRUE, TRUE,
TRUE, FALSE, FALSE, 0}

Status = FT_EE_Program (Handle, &EEData);
if (Status == OK) {
        // The function was successful, the information in EEData is in the EEPROM
}
else {
        // The function failed. The error code can be reviewed and appropriate corrective action taken
}
```

FT_EraseEE (Handle)

This function will erase the EEPROM. If the function executes successfully then it will return OK otherwise it will return an error code.

Visual Basic

Parameter	Type	Description
Handle	long	A number that uniquely identifies the device

C

Parameter	Type	Description
Handle	pvoid	A pointer to a number that uniquely identifies the device

Examples

The following Visual Basic code demonstrates this function.

```
Dim lngHandle As Long
Dim lngStatus As Long

lngStatus = FT_EraseEE (lngHandle)
If lngStatus = OK Then
        'The function was successful, the EEPROM has been erased
Else
        'The function failed. The error code can be reviewed and appropriate corrective action taken
End If
```

The following C code demonstrates this function.

```
PVOID Handle;
ULONG Status;

Status = FT_EraseEE (Handle);
if (Status == OK) {
        // The function was successful, the EEPROM has been erased
}
else {
        // The function failed. The error code can be reviewed and appropriate corrective action taken
}
```

Appendix A
QS Series Visual Basic Header File

This appendix contains the Visual Basic header file that contains all of the function and constant definitions covered in this guide. This text can be copied and pasted into a module in the user's Visual Basic project.

```
'****************************************************************************************************************************************************
' Function declarations
'****************************************************************************************************************************************************
Public Declare Function FT_ListDevices Lib "FTD2XX.DLL" (ByVal arg1 As Long, ByVal arg2 As String, ByVal dwFlags As Long) As Long
Public Declare Function FT_Open Lib "FTD2XX.DLL" (ByVal intDeviceNumber As Integer, ByRef lngHandle As Long) As Long
Public Declare Function FT_OpenEx Lib "FTD2XX.DLL" (ByVal arg1 As String, ByVal arg2 As Long, ByRef lngHandle As Long) As Long
Public Declare Function FT_Close Lib "FTD2XX.DLL" (ByVal lngHandle As Long) As Long
Public Declare Function FT_Read Lib "FTD2XX.DLL" (ByVal lngHandle As Long, ByVal lpszBuffer As String, ByVal lngBufferSize As Long,_
  ByRef lngBytesReturned As Long) As Long
Public Declare Function FT_Write Lib "FTD2XX.DLL" (ByVal lngHandle As Long, ByVal lpszBuffer As String, ByVal lngBufferSize As Long,_
  ByRef lngBytesWritten As Long) As Long
Public Declare Function FT_SetBaudRate Lib "FTD2XX.DLL" (ByVal lngHandle As Long, ByVal lngBaudRate As Long) As Long
Public Declare Function FT_SetDataCharacteristics Lib "FTD2XX.DLL" (ByVal lngHandle As Long, ByVal byWordLength As Byte,_
  ByVal byStopBits As Byte, ByVal byParity As Byte) As Long
Public Declare Function FT_SetFlowControl Lib "FTD2XX.DLL" (ByVal lngHandle As Long, ByVal intFlowControl As Integer, ByVal byXonChar As Byte,_
  ByVal byXoffChar As Byte) As Long
Public Declare Function FT_SetDtr Lib "FTD2XX.DLL" (ByVal lngHandle As Long) As Long
Public Declare Function FT_ClrDtr Lib "FTD2XX.DLL" (ByVal lngHandle As Long) As Long
Public Declare Function FT_SetRts Lib "FTD2XX.DLL" (ByVal lngHandle As Long) As Long
Public Declare Function FT_ClrRts Lib "FTD2XX.DLL" (ByVal lngHandle As Long) As Long
Public Declare Function FT_GetModemStatus Lib "FTD2XX.DLL" (ByVal lngHandle As Long, ByRef lngModemStatus As Long) As Long
Public Declare Function FT_SetChars Lib "FTD2XX.DLL" (ByVal lngHandle As Long, ByVal byEventChar As Byte, ByVal byEventCharEnabled As Byte,_
  ByVal byErrorChar As Byte, ByVal byErrorCharEnabled As Byte) As Long
Public Declare Function FT_Purge Lib "FTD2XX.DLL" (ByVal lngHandle As Long, ByVal lngMask As Long) As Long
Public Declare Function FT_SetTimeouts Lib "FTD2XX.DLL" (ByVal lngHandle As Long, ByVal lngReadTimeout As Long,_
  ByVal lngWriteTimeout As Long) As Long
Public Declare Function FT_GetQueueStatus Lib "FTD2XX.DLL" (ByVal lngHandle As Long, ByRef lngRXBytes As Long) As Long
Public Declare Function FT_SetBreakOn Lib "FTD2XX.DLL" (ByVal lngHandle As Long) As Long
Public Declare Function FT_SetBreakOff Lib "FTD2XX.DLL" (ByVal lngHandle As Long) As Long
Public Declare Function FT_GetStatus Lib "FTD2XX.DLL" (ByVal lngHandle As Long, ByRef lngRXBytes As Long, ByRef lngTXBytes As Long,_
  ByRef lngEventsDWord As Long) As Long
Public Declare Function FT_ResetDevice Lib "FTD2XX.DLL" (ByVal lngHandle As Long) As Long

' New Functions
Public Declare Function FT_SetEventNotification Lib "FTD2XX.DLL" (ByVal lngHandle As Long, ByVal dwEventMask As Long, ByVal Arg As Long)_
  As Long
Public Declare Function FT_ResetPort Lib "FTD2XX.DLL" (ByVal lngHandle As Long) As Long
Public Declare Function FT_RestartInTask Lib "FTD2XX.DLL" (ByVal lngHandle As Long) As Long
Public Declare Function FT_StopInTask Lib "FTD2XX.DLL" (ByVal lngHandle As Long) As Long
Public Declare Function FT_SetResetPipeRetryCount Lib "FTD2XX.DLL" (ByVal lngHandle As Long, ByVal lngCount As Long) As Long

'****************************************************************************************************************************************************
' EEPROM programming function declarations
'****************************************************************************************************************************************************
Public Declare Function FT_EE_Program Lib "FTD2XX.DLL" (ByVal lngHandle As Long, ByRef lpData As PROGRAM_DATA) As Long
Public Declare Function FT_EE_Read Lib "FTD2XX.DLL" (ByVal lngHandle As Long, ByRef lpData As PROGRAM_DATA) As Long
Public Declare Function FT_EE_UARead Lib "FTD2XX.DLL" (ByVal lngHandle As Long, ByVal pucData As String, ByVal dwDataLen As Long,_
  ByRef lpdwBytesRead As Long) As Long
Public Declare Function FT_EE_UAWrite Lib "FTD2XX.DLL" (ByVal lngHandle As Long, ByVal pucData As String, ByVal dwDataLen As Long) As Long
Public Declare Function FT_EE_UASize Lib "FTD2XX.DLL" (ByVal lngHandle As Long, ByRef lpdwSize As Long) As Long
Public Declare Function FT_EraseEE Lib "FTD2XX.DLL" (ByVal lngHandle As Long) As Long

'****************************************************************************************************************************************************
' Supporting functions for conversion from C to Visual Basic
```

```
'*****************************************************************************************************************************************
' Used instead of FT_ListDevices to get the number of devices on the bus
Public Declare Function FT_GetNumDevices Lib "FTD2XX.DLL" Alias "FT_ListDevices" (ByRef arg1 As Long, ByVal arg2 As String,_
    ByVal dwFlags As Long) As Long

'*****************************************************************************************************************************************
' Constant Declarations
'*****************************************************************************************************************************************
' Return codes
Public Const OK = 0
Public Const INVALID_HANDLE = 1
Public Const DEVICE_NOT_FOUND = 2
Public Const DEVICE_NOT_OPENED = 3
Public Const IO_ERROR = 4
Public Const INSUFFICIENT_RESOURCES = 5
Public Const INVALID_PARAMETER = 6
Public Const INVALID_BAUD_RATE = 7

Public Const DEVICE_NOT_OPENED_FOR_ERASE = 8
Public Const DEVICE_NOT_OPENED_FOR_WRITE = 9
Public Const FAILED_TO_WRITE_DEVICE = 10
Public Const EEPROM_READ_FAILED = 11
Public Const EEPROM_WRITE_FAILED = 12
Public Const EEPROM_ERASE_FAILED = 13
Public Const EEPROM_NOT_PRESENT = 14
Public Const EEPROM_NOT_PROGRAMMED = 15
Public Const INVALID_ARGS = 16
Public Const OTHER_ERROR = 17

' Flow Control
Public Const FLOW_NONE = &H0
Public Const FLOW_RTS_CTS = &H100
Public Const FLOW_DTR_DSR = &H200
Public Const FLOW_XON_XOFF = &H400

' Purge rx and tx buffers
Public Const PURGE_RX = 1
Public Const PURGE_TX = 2

' Flags for FT_OpenEx
Public Const OPEN_BY_SERIAL_NUMBER = 1
Public Const OPEN_BY_DESCRIPTION = 2

' Flags for FT_ListDevices
Public Const LIST_BY_NUMBER_ONLY = &H80000000
Public Const LIST_BY_INDEX = &H40000000
Public Const LIST_ALL = &H20000000

' Modem Status
Public Const MODEM_STATUS_CTS = &H10
Public Const MODEM_STATUS_DSR = &H20
Public Const MODEM_STATUS_RI = &H40
Public Const MODEM_STATUS_DCD = &H80

' Event Masks
Public Const EVENT_RXCHAR = 1
Public Const EVENT_MODEM_STATUS = 2

' Baud Rates
Public Const BAUD_300 = 300
Public Const BAUD_600 = 600
```

```
Public Const BAUD_1200 = 1200
Public Const BAUD_2400 = 2400
Public Const BAUD_4800 = 4800
Public Const BAUD_9600 = 9600
Public Const BAUD_14400 = 14400
Public Const BAUD_19200 = 19200
Public Const BAUD_38400 = 38400
Public Const BAUD_57600 = 57600
Public Const BAUD_115200 = 115200
Public Const BAUD_230400 = 230400
Public Const BAUD_460800 = 460800
Public Const BAUD_921600 = 921600

' Word Lengths
Public Const BITS_8 = 8
Public Const BITS_7 = 7
Public Const BITS_6 = 6
Public Const BITS_5 = 5

' Stop Bits
Public Const STOP_BITS_1 = 0
Public Const STOP_BITS_1_5 = 1
Public Const STOP_BITS_2 = 2

' Parity
Public Const PARITY_NONE = 0
Public Const PARITY_ODD = 1
Public Const PARITY_EVEN = 2
Public Const PARITY_MARK = 3
Public Const PARITY_SPACE = 4

' Type declaration for EEPROM programming
Public Type PROGRAM_DATA
    VendorId As Integer             '0x0403
    ProductId As Integer            '0xF448
    Manufacturer As Long            '32 "Linx Technologies"
    ManufacturerId As Long          '16 "LT"
    Description As Long             '64 "LINX SDM-USB-QS-S"
    SerialNumber As Long            '16 "LT000001" if fixed, or NULL
    MaxPower As Integer             '0 < MaxPower <= 500
    PNP As Integer                  '0 = disabled, 1 = enabled
    SelfPowered As Integer          '0 = bus powered, 1 = self powered
    RemoteWakeup As Integer         '0 = not capable, 1 = capable
    ' Rev4 extensions:
    Rev4 As Byte                    'true if Rev4 chip, false otherwise
    IsoIn As Byte                   'true if in endpoint is isochronous
    IsoOut As Byte                  'true if out endpoint is isochronous
    PullDownEnable As Byte          'true if pull down enabled
    SerNumEnable As Byte            'true if serial number to be used
    USBVersionEnable As Byte        'true if chip uses USBVersion
    USBVersion As Integer           'BCD (0x0200 => USB2)
End Type
```

Appendix B
QS Series C Header File

This appendix contains the C header file that contains all of the function and constant definitions covered in this guide. This text can be copied and pasted into a module in the user's C project.

```
#ifndef FTD2XX_H
#define FTD2XX_H

// The following ifdef block is the standard way of creating macros
// which make exporting from a DLL simpler.  All files within this DLL
// are compiled with the FTD2XX_EXPORTS symbol defined on the command line.
// This symbol should not be defined on any project that uses this DLL.
// This way any other project whose source files include this file see
// FTD2XX_API functions as being imported from a DLL, whereas this DLL
// sees symbols defined with this macro as being exported.

#ifdef FTD2XX_EXPORTS
#define FTD2XX_API __declspec(dllexport)
#else
#define FTD2XX_API __declspec(dllimport)
#endif

typedef PVOID       HANDLE;
typedef ULONG       STATUS;

//****************************************************************************************************************************************
//Function declarations
//****************************************************************************************************************************************
FTD2XX_API STATUS WINAPI FT_ListDevices (PVOID pArg1, PVOID pArg2, DWORD Flags);
FTD2XX_API STATUS WINAPI FT_Open (int deviceNumber, HANDLE *pHandle);
FTD2XX_API STATUS WINAPI FT_OpenEx (PVOID pArg1, DWORD Flags, HANDLE *pHandle);
FTD2XX_API STATUS WINAPI FT_Close (HANDLE Handle);
FTD2XX_API STATUS WINAPI FT_Read (HANDLE Handle, LPVOID lpBuffer, DWORD nBufferSize, LPDWORD lpBytesReturned);
FTD2XX_API STATUS WINAPI FT_Write (HANDLE Handle, LPVOID lpBuffer, DWORD nBufferSize, LPDWORD lpBytesWritten);
FTD2XX_API STATUS WINAPI FT_SetBaudRate (HANDLE Handle, ULONG BaudRate);
FTD2XX_API STATUS WINAPI FT_SetDataCharacteristics (HANDLE Handle, UCHAR WordLength, UCHAR StopBits, UCHAR Parity);
FTD2XX_API STATUS WINAPI FT_SetFlowControl (HANDLE Handle, USHORT FlowControl, UCHAR XonChar, UCHAR XoffChar);
FTD2XX_API STATUS WINAPI FT_SetDtr (HANDLE Handle);
FTD2XX_API STATUS WINAPI FT_ClrDtr (HANDLE Handle);
FTD2XX_API STATUS WINAPI FT_SetRts (HANDLE Handle);
FTD2XX_API STATUS WINAPI FT_ClrRts (HANDLE Handle);
FTD2XX_API STATUS WINAPI FT_GetModemStatus (HANDLE Handle, ULONG *pModemStatus);
FTD2XX_API STATUS WINAPI FT_SetChars (HANDLE Handle, UCHAR EventChar, UCHAR EventCharEnabled, UCHAR ErrorChar, UCHAR
    ErrorCharEnabled);
FTD2XX_API STATUS WINAPI FT_Purge (HANDLE Handle, ULONG Mask);
FTD2XX_API STATUS WINAPI FT_SetTimeouts (HANDLE Handle, ULONG ReadTimeout, ULONG WriteTimeout);
FTD2XX_API STATUS WINAPI FT_GetQueueStatus (HANDLE Handle, DWORD *dwRxBytes);
FTD2XX_API STATUS WINAPI FT_SetBreakOn (HANDLE Handle);
FTD2XX_API STATUS WINAPI FT_SetBreakOff (HANDLE Handle);
FTD2XX_API STATUS WINAPI FT_GetStatus (HANDLE Handle, DWORD *dwRxBytes, DWORD *dwTxBytes, DWORD *dwEventDWord);
FTD2XX_API STATUS WINAPI FT_ResetDevice (HANDLE Handle);

FTD2XX_API STATUS WINAPI FT_SetEventNotification (HANDLE Handle, DWORD Mask, PVOID Param);
FTD2XX_API STATUS WINAPI FT_ResetPort (HANDLE Handle);
FTD2XX_API STATUS WINAPI FT_RestartInTask (HANDLE Handle);
FTD2XX_API STATUS WINAPI FT_StopInTask (HANDLE Handle);
FTD2XX_API STATUS WINAPI FT_SetResetPipeRetryCount (HANDLE Handle, DWORD dwCount);

//****************************************************************************************************************************************
//EEPROM function declarations
```

```
//*************************************************************************************************************************************************
FTD2XX_API STATUS WINAPI FT_EE_Program (HANDLE Handle, PPROGRAM_DATA pData);
FTD2XX_API STATUS WINAPI FT_EE_Read (HANDLE Handle, PPROGRAM_DATA pData);
FTD2XX_API STATUS WINAPI FT_EE_UARead (HANDLE Handle, PUCHAR pucData, DWORD dwDataLen, LPDWORD lpdwBytesRead);
FTD2XX_API STATUS WINAPI FT_EE_UAWrite (HANDLE Handle, PUCHAR pucData, DWORD dwDataLen);
FTD2XX_API STATUS WINAPI FT_EE_UASize (HANDLE Handle, LPDWORD lpdwSize);
FTD2XX_API STATUS WINAPI FT_EraseEE (HANDLE Handle);

//*************************************************************************************************************************************************
//Constant Declarations
//*************************************************************************************************************************************************
'//Return codes
enum {
    OK,
    INVALID_HANDLE,
    DEVICE_NOT_FOUND,
    DEVICE_NOT_OPENED,
    IO_ERROR,
    INSUFFICIENT_RESOURCES,
    INVALID_PARAMETER,
    INVALID_BAUD_RATE,

    DEVICE_NOT_OPENED_FOR_ERASE,
    DEVICE_NOT_OPENED_FOR_WRITE,
    FAILED_TO_WRITE_DEVICE,
    EEPROM_READ_FAILED,
    EEPROM_WRITE_FAILED,
    EEPROM_ERASE_FAILED,
    EEPROM_NOT_PRESENT,
    EEPROM_NOT_PROGRAMMED,
    INVALID_ARGS,
    OTHER_ERROR
};

// Flow Control
#define FLOW_NONE       0x0000
#define FLOW_RTS_CTS    0x0100
#define FLOW_DTR_DSR    0x0200
#define FLOW_XON_XOFF           0x0400

// Purge rx and tx buffers
#define PURGE_RX        1
#define PURGE_TX        2

// FT_OpenEx Flags
#define OPEN_BY_SERIAL_NUMBER           1
#define OPEN_BY_DESCRIPTION             2

// FT_ListDevices Flags (used in conjunction with FT_OpenEx Flags)
#define LIST_NUMBER_ONLY        0x80000000
#define LIST_BY_INDEX           0x40000000
#define LIST_ALL                0x20000000
#define LIST_MASK (LIST_NUMBER_ONLY | LIST_BY_INDEX | LIST_ALL)

// Modem Status
#define MODEM_STATUS_CTS        &H10
#define MODEM_STATUS_DSR        &H20
#define MODEM_STATUS_RI         &H40
#define MODEM_STATUS_DCD        &H80

// Event Masks
```

```
#define EVENT_RXCHAR            1
#define EVENT_MODEM_STATUS      2

// Baud Rates
#define BAUD_300                300
#define BAUD_600                600
#define BAUD_1200               1200
#define BAUD_2400               2400
#define BAUD_4800               4800
#define BAUD_9600               9600
#define BAUD_14400              14400
#define BAUD_19200              19200
#define BAUD_38400              38400
#define BAUD_57600              57600
#define BAUD_115200             115200
#define BAUD_230400             230400
#define BAUD_460800             460800
#define BAUD_921600             921600

// Word Lengths
#define BITS_8                  (UCHAR) 8
#define BITS_7                  (UCHAR) 7
#define BITS_6                  (UCHAR) 6
#define BITS_5                  (UCHAR) 5

// Stop Bits
#define STOP_BITS_1             (UCHAR) 0
#define STOP_BITS_1_5           (UCHAR) 1
#define STOP_BITS_2             (UCHAR) 2

// Parity
#define PARITY_NONE             (UCHAR) 0
#define PARITY_ODD              (UCHAR) 1
#define PARITY_EVEN             (UCHAR) 2
#define PARITY_MARK             (UCHAR) 3
#define PARITY_SPACE            (UCHAR) 4

// Type declaration for EEPROM programming
typedef struct PROGRAM_DATA {
        WORD VendorId;                  // 0x0403
        WORD ProductId;                 // 0xF448
        char *Manufacturer;             // 32, "Linx Technologies"
        char *ManufacturerId;           // 16, "LT"
        char *Description;              // 64 "LINX SDM-USB-QS-S"
        char *SerialNumber;             // 16 "LT000001" if fixed, or NULL
        WORD MaxPower;                  // 0 < MaxPower <= 500
        WORD PnP;                       // 0 = disabled, 1 = enabled
        WORD SelfPowered;               // 0 = bus powered, 1 = self powered
        WORD RemoteWakeup;              // 0 = not capable, 1 = capable
        // Rev4 extensions
        UCHAR Rev4;                     // true if Rev4 chip, false otherwise
        UCHAR IsoIn;                    // true if in endpoint is isochronous
        UCHAR IsoOut;                   // true if out endpoint is isochronous
        UCHAR PullDownEnable;           // true if pull down enabled
        UCHAR SerNumEnable;             // true if serial number to be used
        UCHAR USBVersionEnable;         // true if chip uses USBVersion
        WORD USBVersion;                // BCD (0x0200 => USB2)
} PROGRAM_DATA, *PPROGRAM_DATA;
```